OULU UNIVERSITY OF
APPLIED SCIENCES

Jukka Ojala

# TEST TOOL DEVELOPMENT OF SECURE REAL-TIME TRANSPORT PROTOCOL

With analysis of media signaling methods

# TEST TOOL DEVELOPMENT OF SECURE REAL-TIME TRANSPORT PROTOCOL

With analysis of media signaling methods

Jukka Ojala
Master's Thesis
Autumn 2014
Degree Programme In Information Technology
Oulu University of Applied Sciences

# ABSTRACT

Oulu University of Applied Sciences
Master of Engineering in Information Technology

---

Author(s): Jukka Ojala
Title of Master's thesis: Test Tool Development of Secure Real-time Transport Protocol
Supervisor(s): Lauri Piikivi (Codenomicon), Tuomo Tikkanen (OUAS)
Term and year of completion: Autumn 2014 Number of pages: 52 + *3 appendices*

---

Testing of VoIP security problems is essential to enterprises providing real-time data and services. The critical infrastructure can be compromised and data to be regularly exposed. This information may be used by malicious hackers for illegal purposes. At the same time consumers are connecting their smartphones and other personal devices to the network. The business with private and secured data is becoming more and more important in our everyday life.

The major security threat for enterprises today is discovering existing unknown vulnerabilities from used software. Unknown vulnerabilities may cause a lot of serious problems, attacks utilizing security holes may continue undetected (while customer safety is compromised) and the repair process tends to be slow once an attack is detected. Additionally, during maintenance there is downtime when customer services are not accessible. Naturally, all downtime is damaging company reputation and eventually affecting to company profits. Security related bugs are likely to exist in platforms when facing with new technologies which are complex to implement and when software versions are often a released detriment of testing. Also, an increasing number of vulnerabilities have not been published but instead those may end to be used and shared within underground hacker communities. Companies and enterprises need to find preventive actions to protect their products and services because it takes too much time to wait fixed software releases from vendors.

In this thesis work the protocol model from a Secure Real-time Transport Protocol (SRTP) is implemented using the Codenomicon Test Tool platform. The product can send valid and anomalous data transmissions to the test application over the Internet. This is technique (or testing method) called fuzzing or fuzz testing. Software robustness can be verified and unknown or zero-day vulnerabilities may be found with fuzzing.

RTP data is typically audio, video or other streaming content. RTP data (or media stream) is usually transferred and embedded within another protocol. The SRTP Test Suite can utilize other Codenomicon test tools to do media signaling negotiation with the test targets in a way which is secure when it comes to cryptography. Thus, the result of the connection parameters may be used to get connected with a wider variety of test targets and send an SRTP fuzzing data stream to these applications.

---

Keywords:

fuzzing, cryptography, cipher, key derivation, encryption, authentication

# TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Master of Engineering in Information Technology

Tekijä: Jukka Ojala
Opinnäytetyön nimi: Suojatun ja reaaliaikaisen siirtoprotokollan testaustyökalun tuotekehitys
Työn ohjaaja(t): Lauri Piikivi (Codenomicon), Tuomo Tikkanen (OAMK)
Työn valmistumislukukausi ja -vuosi: syksy 2014 Sivumäärä: 52 + *3 liitesivua*

VoIP-järjestelmien turvallisuuden testaaminen on oleellista yrityksille, jotka tuottavat reaaliaikaista tietoa ja palveluita. Kriittinen infrastruktuuri ja salatut tiedot voivat olla jatkuvassa paljastumisvaarassa. Tämä tieto voi olla vihamielisien hakkereiden laittomassa käytössä. Samaan aikaan kuluttajat liikkuvat tietoverkoissa älypuhelimilla ja muilla henkilökohtaisilla laitteilla. Yksityisellä ja salatulla tiedolla tehtävä kaupankäynti ja siihen rinnastettavat tehtävät ovat jokapäiväistä elämää.

Yritysten merkittävä turvallisuusuhka on löytää olemassa olevat tuntemattomat haavoittuvuudet käytetyistä ohjelmistoista. Tuntemattomat haavoittuvuudet voivat aiheuttaa useita vakavia ongelmia, hyökkäykset, jotka hyödyntävät tietoturva-aukkoja, voivat jatkua huomaamatta (samaan aikaan kun asiakasturvallisuus on paljastumisvaarassa) ja korjausprosessit saattavat olla pitkäkestoisia kun hyökkäys on havaittu. Lisäksi kunnossapito on häiriöaikaa, jolloin asiakaspalvelut eivät ole saatavilla. Luonnollisesti edellinen tila vaurioittaa yrityksen mainetta ja lopulta vaikuttaa yrityksen tulokseen. Tietoturvaan liittyvät viat ovat todennäköisiä alustoilla, jotka käyttävät uutta teknologiaa ja joiden toteutus on päätynyt monimutkaiseksi kokonaisuudeksi. Tietoturvallisuus vaarantuu, jos ohjelmistoversiot julkaistaan testauksen kustannuksella. Myös lisääntyvä määrä haavoittuvuuksia jää hakkeriyhteisöjen salaiseksi tiedoksi. Yritysten täytyy löytää ennaltaehkäiseviä keinoja turvata omat tuotteet ja palvelut, koska vie liian kauan aikaa odottaa toimittajan julkaisuja ohjelmistokorjauksista.

Tässä opinnäytetyössä toteutetaan protokollamalli suojatusta reaaliaikaisesta tietoliikenneprotokollasta (SRTP) käyttäen Codenomicon-testaustyökalualustaa. Tuote voi lähettää aitoja ja epätavallisia protokollaviestejä tietoverkon yli testattavalle sovellukselle. Tätä tekniikkaa (tai testausmetodia) kutsutaan nimellä fuzzing-testaus. Ohjelmiston kelpoisuus voidaan varmistaa sekä tuntemattomat ja nollapäivän haavoittuvuudet voidaan löytää fuzzing-testauksella.

RTP-tieto on tyypillisesti ääntä, kuvaa tai muuta suoratoisto sisältöä. Tieto (tai median suoratoisto) siirretään tavallisesti sulautettuna sisältönä osana jotain muuta tiedonsiirtoprotokollaa. SRTP-testaustyökalu voi hyödyntää toista Codenomiconin testaustyökalua tekemään siirrettävälle medialle signalointineuvottelun kryptograafisesti turvallisella tavalla yhdessä testauskohteen kanssa. Siten yhteysparametreja voidaan käyttää yhteydenmuodostukseen laajemman testauskohteiden valikoiman kanssa ja lähettää SRTP fuzzing-testauksen testitapauksia tietovirtana näille sovelluksille.

Asiasanat:
fuzzing-testaus, tietoturva, kryptografia, salakirjoitus, salausmenetelmät, autentikaatio

CONTENTS

# LIST OF ABBREVIATIONS

| | |
|---|---|
| ABNF | Augmented Backus-Naur Form |
| AES | Advanced Encryption Standard |
| BNF | Backus-Naur Form |
| CS | Crypto Session |
| CSB | Crypto Session Bundle |
| DNS | Domain Name System |
| DOS | Denial of Service |
| HMAC | keyed-Hashed Message Authentication Code |
| HTTP | Hypertext Transfer Protocol |
| IETF | Internet Engineering Task Force |
| IV | Initialization Vector |
| MKI | Master Key Identifier |
| MIKEY | Multimedia Internet KEYing |
| PRF | PseudoRandom Function |
| RFC | Request for Comments |
| RIP | Routing Information Protocol |
| ROC | Rollover Counter |
| RTP | Real-time Transport Protocol |
| RTCP | Real-time Control Protocol |
| SDES | SDP Security Descriptions for Media Streams |
| SDP | Session Description Protocol |
| SHA | Secure Hash Algorithm |
| SRTP | Secure Real-time Transport Protocol |
| SIP | Session Initiation Protocol |
| SUT | System Under Test |
| TGK | Traffic Generation Key |
| TEK | Traffic-encryption Key |
| UAS | User Agent Server |
| VoIP | Voice over IP |
| XML | Extensible Markup Language |
| ZRTP | Z and Real-time Transport Protocol |

# 1 INTRODUCTION

A Voice over Internet Protocol (VoIP) is a technique and group of technologies for transferring a voice communications over the Internet Protocol (IP) based networks. The VoIP benefits and attractiveness for enterprises and organizations point of view are the cost-effectiveness, the flexibility and a possibility to utilize the existing network for voice communication. Some enterprises have taken the VoIP in part of their internal communications and also for the customer service. (Takanen & Vuontisjarvi 2011, 2.)

Although VoIP, similar terms like of an *IP Telephony* and *Converged Networks* which all have slightly different definitions, those are often used interchangeably. With these used terms, are referring to the structures and processes from design and implementation of a common networking infrastructure which handles data, voice and multimedia communications. In the first decade of 21th century, main focus was in the voice. The enthusiasm was to replace the circuit-switched voice with the packet-switched voice within the enterprise and at home over network connections. (Porter 2005, 6.)

The unknown vulnerabilities are software defects and bugs not yet to be discovered. While known vulnerabilities are blocked and excluded with the software patches and updates, security holes of the unknown vulnerabilities remain in the system after every software update. Vulnerabilities in applications providing a customer interfaces are commonly used way for a hacker to attack toward an enterprise and steal trusted information. Fixing of the software vulnerabilities in precautionary should be on a high priority. Firewalls and antivirus software does not offer protection for a security holes but increases overall system complexity. An augmented complexity is on the other hand a threat because then the hidden attack surface of the system is increased. The hacker may gain entry to a VPN connected system via another user in a same network which is not aware of the cracked entry. (Takanen & Juuso 2010, 2.)

A cryptographic mechanism plays a crucial role in securing signaling negotiation and media stream connections. The cryptography is a vast field and supports a large number of applications. Some of those (for example a Multimedia Internet KEYing (MIKEY)) are studied in this thesis work. The cryptography is area of the science where a public channel messages are protected

with the security services. These services are *confidentiality*, *integrity protection* and *authentication.*

This thesis work questions/main tasks are:
- produce a new version of the model based fuzzer for the secure rtp (the model and the test suite are fully rewritten to support the current platform generation)
- study of the secure rtp signaling (key exchanges) and implement part of these. Which signaling environments are the most useful for the rtp fuzzing?

Smaller implementation and studying subtasks or goals are:
- SRTP model creation (grammar and model structures)
- SRTP controller creation (Java component)
- AES in Counter mode (Java crypto component)
- MIKEY-PSK method (Java crypto component)

### 1.1 Basic concepts and terms

A short introduction of the essential concepts in this thesis is given in this chapter.

|| denotes concatenation of two variables. For an example, in C = A || B the most significant bits of the C are total bits of the A and the least significant bits are equal to the bits of the B.

*An attribute Grammar* is “an extension of a context-free grammar having attribute associated to the grammar symbol” (Kaksonen 2001, 43)

*The context-Free Grammar* is also called a *Backbus-Naur Form* (BNF). It basically describes the syntax notation of language and the message format of a protocol.

*A Zero-day vulnerability* (or *zero-day attack*) the first sentence is meaning a previously unknown vulnerability in a computer application which has no patch to fix the vulnerability. The second sentence means the attack with utilizes a security hole that is not known by a software developer. Attackers may share zero-day exploits with themselves before the attack against the software.

## 1.2 Fuzzing

Fuzzing is identified as a black box testing, as it does not need any access to a System Under Test (SUT) source code. During a test process, the SUT is exposed to the malformed data and robustness abilities in these conditions are tested.

The core principle of the fuzzing is described in a nutshell:

> "Fuzzing or fuzz testing is a negative software testing method that feeds a program, device or system with malformed and unexpected input data in order to find defects" (Codenomicon 2014, date of retrieval 6.1.2014)

Fuzzing enables software testers and designers to find defects which can be triggered using malformed inputs via the external interfaces. With this ability, fuzzing (or fuzz testing) covers exposed and critical attack surfaces relatively well. It also identifies common software errors and vulnerabilities. (Codenomicon 2014, date of retrieval 6.1.2014)

The software programs which are used to run and create the fuzz testing are called *fuzzers*. Couple real-life examples of these is a tool which sends malformed Hypertext Transfer Protocol (HTTP) requests to a web server or other one which sends Routing Information Protocol (RIP) version 2 routing requests with invalid cryptographic fields.

### 1.2.1 Static vs model based fuzzing

A fuzzer can be commonly categorized to the *static* and the *model-based fuzzers*. The static or non-stateful fuzzer may not do any kind of a security testing after begin point of a protocol. In the contrast, a fully model-based fuzzer is able to penetrate more on the tested protocol and thus leading to be more effective in discovering the security flaws and vulnerabilities. The static fuzzers are missing of the ability to do protocol runtime operations such as length or cryptographic operations and calculations or replacing elements from the incoming messages to the outgoing messages. (Codenomicon 2014, date of retrieval 8.1.2014)

The model based fuzzer design is introduced in chapter 3 in more detail.

## 1.3 Introduction to cryptographis mechanisms

When building a component with the cryptographic services, following services and concepts need to be studied and observed. In generally, a cryptographic algorithm determines the data transformation. Algorithm receives a data block which security services need to be provided with a key as input parameters. The data blocks are fixed-length sequences of the bits (data may be padded with the zero bits to get the data block of desired length). An algorithms based on the key type can be divided on the *symmetric* or the *public (or asymmetric) key cryptography*. To produce secure communication in the symmetric key cryptography each parties need to agree on a shared key. The encryption and decryption keys are usually identical and can be derived from shared key. This concept is named as a *cipher* or *ciphering*. The ciphers are divided to a *block* and a *stream ciphers* and internally for both: the functionality or the cryptographic mode. Modes are used to add Boolean XOR operations and the feedback additions to the basic ciphering calculations. As a result of these additions, the different ciphertext output is generated in each round of the calculation from identical plaintext input. On the asymmetric algorithm, key pairs (private and public key) are used in the public key cryptography. A private key is used for the encryption and public key for decryption. A public key can be derived from a private key and not usable in vice versa. (Sisalem et al. 2009, 6)

The trusted and reliable security needs following cryptographic services. Firstly, the *confidentiality* consists of two inverses processes, called an encryption and a decryption. The encryption of data enables a sender of data to transform a *plaintext* message (which is for example our normal written text) into the *ciphertext* using a secret key. A data in the ciphertext format seems to be randomized sequence of the bits when viewed by a third party and it's not possible to do inverse operation of the encryption (decryption) for recovering data back to the plaintext without using a key and proper cipher algorithm. Because of the cipher specifications, algorithms and modes are public documented, the mode security depends on the secrecy of the key. Secondly, the *integrity protection* enables a recipient to ensure that the received message content is the original and a packet has not been tampered during path of a transmission. Any kind of changes to the data causes failure in the data integrity. Thirdly, the data *authentication* enables the recipient can rely and confirm that the received message is sent from the originator entity who was initially declared to be sender. (sp800-38a_Block_Cipher_Modes; Sisalem et al. 2009, 5)

### 1.3.1 Block and stream ciphers

A block cipher is used to transform one block of a plaintext (P) to corresponding block size of a ciphertext (C) using a shared (secret) key (K). Inverse operation recovers a plaintext from a ciphertext.

*FORMULA 1. Encryption and decryption operations of a block cipher*

$C_i = E(P_i)\ respectively\ P_i = D(C_i)$ where

E = Encryption

D = Decryption

i = Block index

A stream cipher is used to perform the bit-XOR operations between an input stream and a keystream of the same length. Due to these operations those are called an additive ciphers. The key generator produces the keystream which must be a pseudorandom stream of bits. The stream cipher implementation may utilize the block ciphers and the cryptographic modes. Then the keystream is generated by the block cipher. (Sisalem et al. 2009, 7)

*FORMULA 2. Encryption and decryption operations of a stream cipher*

$C_i = P_i \oplus K_i\ respectively\ P_i = C_i \oplus K_i$ where

C = Ciphertext

P = Plaintext

K = Keystream

$\oplus$ = XOR operation

i = Bit index

Advanced Encryption Standard (AES) is well-known a symmetric block cipher algorithm. The AES uses a data block or sequences of 128 bits (digits with values of 0 or 1) as an input block and produces same size of an output block as a result. The cipher may be initialized with the key lengths of 128, 198 or 256 bits. (FIPS AES 2001)

Some modes of the operation are requiring a random data block called the Initialization Vector (IV) as an additional input block. Depending on the operation mode the input block (plaintext or

ciphertext) and IV can be XOR-ed together. The IV has to be unpredictable for the Cipher Block Chaining (CBC) and the Cipher Feedback (CFB) modes and unique (*nonce, a value that is used only once) for the Output Feedback (OFB) and the Counter (CTR) modes for each execution round of encryption process to maintain cryptographic message confidentiality. The IV does not need to be secret and IV itself or calculation information of the IV can be transferred with the ciphertext. For the CTR mode of operation an increasing counter value (nonce) is added to the IV. (Dworkin 2001)

The CTR mode is synchronous stream cipher where a keystream generation is an independent functionality. When the block cipher is used to generate keystream then the keystream is multiple of the block size. Possible leftover data of the plaintext or the ciphertext is discarded. Pros of the stream ciphers for encrypting real-time media are no padding is needed, sender can pre-compute the keystream and confidentiality operations can be run in parallel. Con side is that the keystream must not be used more than once or otherwise the plaintext (and also the keystream when ciphertext and plaintext are XOR-ed) is easy to break after using of the ciphertext produced by same keystream. The keystream reuse is called a “two-time pad”. The stream ciphers are also vulnerable to the bit flipping, if an attacker knows of the plaintext then he/she can alter the ciphertext so that the end result of plaintext has corrupted. To protect these effects, a Message Authentication Code (MAC) must be used. Following *FIGURE 1. Counter (CTR) Mode* describes CTR mode of an operation. The Initialization Vector (IV) is combination of reused nonce and counter value making the IV as unique input parameter for each round of the encryption. The keystream is encryption result of the IV and a key value. The ciphertext is generated by the XOR operation with corresponding keystream and plaintext. In decryption, a plaintext is recovered after the XOR operation with a keystream and a ciphertext. (Sisalem et al. 2009, 9-11)

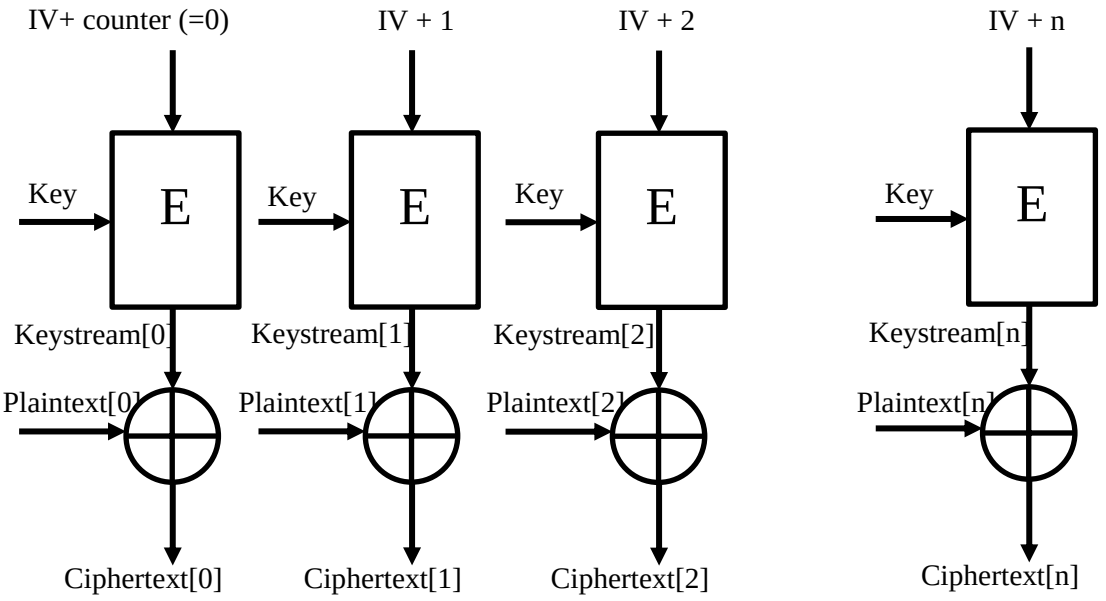



*FIGURE 1. Counter (CTR) Mode*

### 1.3.2 Pseudorandom functions

A pseudorandom function (PRF) is used to produce output (a secret key) value that can not be distinguished from a random data. A key establishment protocol may be used to create a shared secret between two parties. An additional session keying material is generated from the shared secret key. A key material is necessary because cryptographic operations are requiring number of session keys and longer keystreams than length of the shared secret. The session keys are used to protect protocol signaling and transmission of a media data. The keys need to be distinct for session parties and may need to be deviant for each data streams within one session. The PRF applies recursively a keyed-Hashed Message Authentication Code (HMAC) or a cipher to produce mathematically strong keying material. The PRF uses a secret key as it operational input with a block or sequence of random numbers with defined constant values to ensure produced session keys are distinct. (Sisalem et al. 2009, 11)

### 1.3.3 Cryptographic hash functions

A cryptographic hash function takes message as input and produces a digest which is a fixed size result of mathematical transformation. (with a some hash function calculation deviations digest may be called also *a message digest*, *a hash value* or *digital fingerprint*). In a cryptographic func-

tion family a Secure Hash Algorithms (SHA) and a Message-Digest Algorithm (MD5) are widely used. The MD5 produces a 128 bit length of digest. There are known collision and password cracking vulnerabilities of the MD5 hash functions. The most widely used by applications and protocols of the SHA family is SHA-1 but it contains known (theoretical) cryptographic weakness. The SHA-1 has seen as successor of MD5 and it produces a 160 bit length of digest. Integrity protection of cryptographic algorithm is enabled by calculating the hash value over the message and a secret key value. This concept is called the HMAC presented in previous chapter. (Sisalem et al. 2009, 20-21; Stevens 2012)

### 1.3.4 Threats and vulnerabilities

To the produce secure communication following components or concepts are required:

- Mutual authentication for determine presented credentials
- Key exchange for obtain shared secret
- Symmetric cryptographic algorithms for integrity-protected and confidential communication

An attacker aims to break trusted communication via deciphering confidential messages, to inject manipulated data into the communication channel, replay messages with valid or invalid content or in generally tries to disturb a data exchange between two parties. To get access of a plaintext messages an attacker have to acquire the secret key which was used to encrypt the data. There are number of alternatives to achieve this. (Sisalem et al. 2009, 38-39)

If a cryptographic algorithm is tried to break, a ciphertext can be used to mount a *ciphertext-only attack*. It's basic form is a *brute force attack*. In the brute force attack form, all possible keys are tried systemically. This type of an attack is useful for relatively small key sizes because it may require use all of the possible key values until correct one is found. The attacker may also exploit weakness of the cryptographic algorithm, for example of a weak key. The weak keying makes the cipher work predictable ways. Use of a random value in encryption is efficient for preventing predictive nature of the used encryption key. If sections or linking between a plaintext and a ciphertext are known, it may lead to a *known-plaintext attack*. The message headers and padding are in generally public or at least easily to guess. This attack form is more efficient if a manipulated plaintext can targeted to specific one which then yields to a *chosen-plaintext attack*. One use case of the latter is the *differential cryptanalysis* where pairs of plaintexts are producing a constant difference (a statistical and frequent pattern to ciphertext). This may be exploited to deter-

mine the most probable keys to for encryption. The attacker may also detect exploit of how cryptographic algorithm is used, the keystream reuse is most known use case in this category. The implementation of the cryptographic algorithm may reveal valuable information to a skillful attacker. One use case is a *timing attack* which can reveal used algorithm and furthermore lead the breaking of used key by measuring time to decrypt a group of known ciphertexts. For protection to the timing attack the cipher processing can be blurred with additional delay time in range of tolerating decreased performance. (Sisalem et al. 2009, 39)

Protection against replay attacks can be achieved by adding randomness in each round of message exchange. In a Denial of Service (DOS) type of attack, a person is disturbing the communication between two parties but not necessary accessing the private information. The key exchange protocols are also vulnerable to the attacks. One use case is a *man-in-the-middle-attack* where an attacker is located between two parties and accessible to the keying material exchanged. Latter can be mount to a *bid-down-attack* where attacker interferes the session negotiation in such way that target of the attack chooses security mechanism which the attacker can break more easily. Protection against these types of the attacks can be achieved by mutual authentication and with integrity protection in the key exchange. An attacker may also combine latter and DOS attacks to key negotiation to slow down or crash target by this way. (Sisalem et al. 2009, 39-40)

It may seem that the security issue is not a big problem because the cryptography services are providing to numerous if not all the security problems. In the every-day practice these problems are real. The main reasons are: security mechanisms are not fully deployed or are not used at all, components using their own security mechanisms are not interoperable with each other (leading to complementing mutual services or causing security holes) or protocols which require involvement of intermediate entities may lead to communication participants have to trust third parties in the security issues. In a practice, an architectural complexity is frequently causing security vulnerabilities. In this category typical use cases are introduced. Firstly, when a key exchange protocol is piggy-backed to signaling protocol. Examples of the first protocols are SDES and MIKEY and latter SIP. In this case a media plane security depends on security of a signaling plane. Secondly, when signaling plane and media plane are operating independently. In this case, a cryptographic binding between places is needed. Examples of the media plane key exchange protocols are ZRTP and DTLS-SRTP. (Sisalem et al. 2009, 40-41)

# 2 VOIP COMMUNICATION AND MEDIA SECURITY

The VoIP is commonly used term of delivering voice communications and multimedia sessions over the IP in today's conversation and technical (also commercial) articles. Term is not referring to any particular protocol specification but rather to the protocol family used co-operatively together. Each protocol has specific functions and responsibilities such as the call creation, the signaling or media data transfer. The signaling protocols are used to establish and control call sessions. The keying protocol's task is to perform key exchange and negotiate the cryptographic parameters. Media transfer protocol's task is to transfer video and audio data packets between participants of (media) session. Mainly these protocols are based on open standards (like Session Initiation Protocol (SIP), H.323 or RTP) and others are based on proprietary standards (like Skype). Transferred packets are delivered either on a Transport Control Protocol (TCP) for reliable transport services or on User Datagram Protocol (UDP) for quick deliveries of packets with minimal overhead and delay. The TCP packet delivery is distinguishingly connection based and the UDP has connectionless packet delivery (which means it can't guarantee reliable packet delivery).

The VoIP (media) packets are most often transferred on UDP. The RTP tasks are to transfer voice and video data packets with offering fast end-to-end packet delivery. Because of the UDP is distinguishingly connectionless in delivering packets, the RTP is most commonly located on top of the UDP. Characteristics of the RTP is introduced in chapter 2.2 Media Transfer and Security Issues and it's subchapters.

The SIP has increasingly gained in popularity and it has become de-facto standard in a VoIP signaling. The SIP also has some competing protocols, such as H.323, MGCP/Megaco and Skype. First was greatest rival in the 1990s with key difference in binary encoding allowing better performance than SIP (which is text encoded protocol). The strengths in text encoding are easier debugging, doing of modifications and integration to applications. The second competitor is based on traditional telephony approach. This is causing that new features need to introduce simultaneously in telephone and in network. With the SIP it is enough that the telephones agree on services and changes to the network are rarely needed. MGCP/Megago-based VoIP development exists today in France. Skype, as the last rival is proprietary standard. It's secured against common security threats and reverse engineering. A closed environment is also a great weakness

because technology companies prefer to support open standards and environments instead. While VoIP gets more popular, it attracts also illegitimate users and hackers to interference and to intrude call and media sessions. For the VoIP to succeed in the long term, services must offer and guarantee the security and protection to the user. (Sisalem, Floroiu, Kuthan, Abend & Schulzrinne 2009, 2, 44)

Security threats with some example qualities can be commonly categorized as follows:

- **Social threats,** containing unsolicited calls, privacy, identity theft and fraud.
- **Eavesdropping,** containing monitoring of signaling and media information
- **Interception and modification,** containing intercepting exchanged eavesdropping data or with access to the VoIP service components, an intruder can reroute and block calls or degrade of call quality.
- **Service abuse,** containing bypassing a provider authentication, stealing the service (from user) or launching attacks to users or service providers.
- **Interruption of service,** containing DOS attacks aiming to make service unavailable to users. (Sisalem et al. 2009, 3)

Call creation and signaling with the SIP is introduced next.

## 2.1 Call creation and signaling with SIP

The SIP standard has been designed to allow participating devices or applications to setup and teardown (ie terminate active call session) a multimedia sessions such as a voice calls over the Internet. The voice calls can be created between two or more participants. An operational behavior model has been inherited from the email world. User belongs on to own administrative domain. User's software application communicates with each other inside and outside of their administrative domains. Domains are identified by Domain Name System (DNS) names which are shown in latter part of users' addresses. Abstract network topology called *SIP trapezoid* is presented in *FIGURE 2. The SIP Trapezoid* (Sisalem et al. 2009, 43, 49)

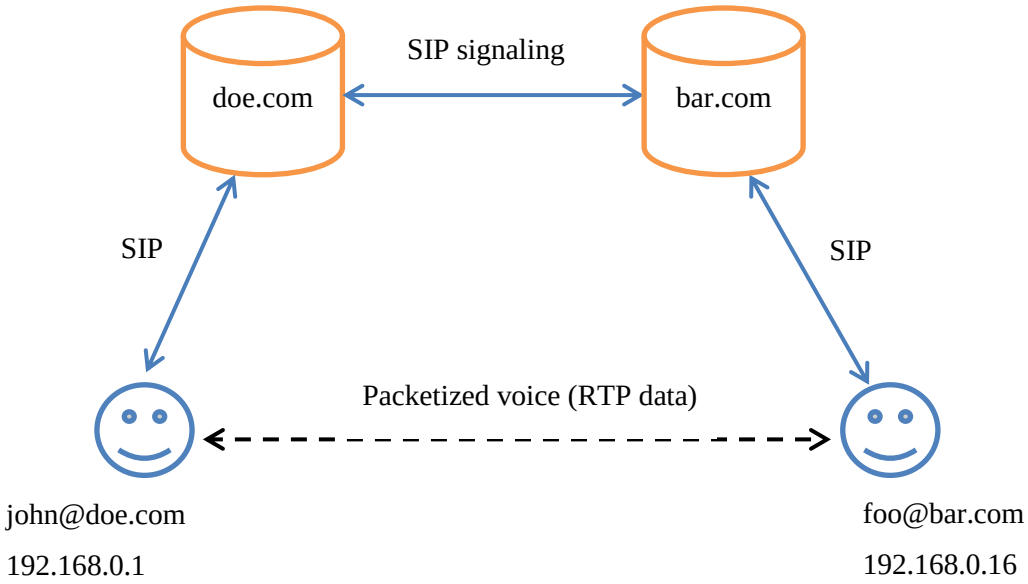



*FIGURE 2. The SIP Trapezoid*

Human faces on bottom corners of the trapezoid are representing end-users (ie SIP telephones or softphones which human users are using to talk each other). Signaling path is represented by softphones communicating inside their domain with the SIP servers (located on top corners) and further SIP servers communicating between domains. Packetized voice data transferred by the RTP is represented on dashed line. Prerequisite condition for this packet transfer is that phones have found each other and agreed of voice session and data encoding. (Sisalem et al. 2009, 49)

### 2.1.1 Session definition and configuration

In generally when initiating a VoIP calls or media streaming there is need to transfer media parameters such as transport address, media port and codec information. The session parameters are described in session invitation with a Session Description Protocol (SDP). The SDP may be used as protocol payload with appropriate transport protocol such as the SIP. Carrying of the session descriptions allow two participants to agree (ie negotiate) on a set of compatible media types. (Handley, Jacobson, Perkins, date of retrieval 16.3.2014)

Session description in the SDP format includes information of: media type, media format, transport protocol, the IP address for media and port for media. Session parameters are defined, by <type>=<value> notation, on session level followed by zero or more media level definitions.

Session level starts from “v=” parameter and continues until first media level parameter which is “m=”. Session level definitions are valid for entire session unless overwritten by equivalent media level parameter. (Handley et al., date of retrieval 16.3.2014)

Example of a SDP payload is shown below

```
v=0
o=jdoe 2890844526 2890842807 IN IP4 10.47.16.5
s=SDP Seminar
i=A Seminar on the session description protocol
u=http://www.example.com/seminars/sdp.pdf
e=j.doe@example.com (Jane Doe)
c=IN IP4 224.2.17.12/127
t=2873397496 2873404696
a=recvonly
m=audio 49170 RTP/AVP 0
m=video 51372 RTP/AVP 99
a=rtpmap:99 h263-1998/90000
```

which is explained in details by Handley et al. By RTP point of view the most important session and media level attributes are connection data “c=”, media descriptions “m=” and those attribute definitions “a=”.

### 2.2 Media Transfer and Security Issues

The service providers and the end users are expecting secure architecture functionality that covers also exchanged media traffic in addition to the signaling messages. Feasibility of the securing data for multimedia communications in an end-to-end manner is an important question. There are legal as well as technical perspectives ahead. From the legal side of view, it must be possible to decrypt of the specific data by a third party on the basis of interception warrant and without participation of original parties. From the technical side of view, some middleware boxes need access to do the data processing of unencrypted (plain) data. Therefore, an end-to-middle security model is necessary in such scenarios. (Sisalem et al. 2009, 174)

### 2.2.1 The Real-Time Transport Protocol

The Real-time Transport Protocol (RTP) is the Internet Engineering Task Force (IETF) specification as Request For Comments (RFC) 3550. The latter is commonly utilized for a real-time communication over the Internet. The RTP provides end-to-end audio and video packet delivery services. These services are including the payload identification, sequence numbering, time stamping and delivery monitoring. The RTP is most typically run on top of the UDP. The RTP has no information of a connection and it has no length information. Maximum length of the packets is limited by underlying protocols. The audio and video media packets are transmitted on a separate RTP sessions. (Schulzrinne, Casner, Frederick, Jacobson, date of retrieval 12.1.2014)

The sub-protocol called RTP Control Protocol (RTCP) is defined and included within of the RTP specification. The RTCP monitors quality of service and conveys information of the participants of on-going session. These packets are usually transferred one port number higher than the RTP packet's port. Exception to latter is when the RTP/RTCP packets are multiplexed ie those can be sent from a single port number. When the packets are multiplexed, it need to be informed with the SDP parameter *a=rtcp-mux*. Deviating from the RTP packets, each RTCP packet contains length information in a 32-bit word minus one meaning zero length for packets including only the RTCP header part. (Schulzrinne et al. 2003, date of retrieval 26.11.2014)

The RTP data packet contains following fields (with short field descriptions):

- **Version (V)**: 2 bits, the version of the RTP
- **Padding (P)**: 1 bit, the packet contains one or more additional padding octets if bit is set to 1.
- **Extension (X)**: 1 bit, the packet contains one extension header if bit is set to 1.
- **CSRC Count (CC)**: 4 bits, the number of included CSRC identifiers
- **Marker (M)**: 1 bit, signaling bit of significant events defined by the RTP profile
- **Payload Type (PT)**: 7 bits, identifier of the RTP payload format
- **Sequence Number**: 16 bits, an incremented number for each sent RTP packet
- **Timestamp**: 32 bits, the timestamp value of sampled RTP packet
- **SSRC**: 32 bits, the synchronization source identifier (chosen by random)
- **CSRS list**: 0 to 15 items, 32 bits each, the contributing sources which number is identified by CC field

The RTCP packet *header* contains following fields:

- **Version (V)**: 2 bits, the version of the RTP
- **Padding (P)**: 1 bit, as with the RTP, included either with an individual or a compound RTCP packet. *Note, only the last individual RTCP packet P bit MUST be set to 1 with compound RTCP packet including padding.*
- **Reception Report Count (RC)**: 5 bits, a number of contained RC blocks
- **Packet Type (PT)**: 8 bits, identifies format of the RTCP packet
- **Length**: 16 bits, the length of RTCP packet excluding length of the RTCP 32 bit header (from version until length field itself)
- **SSRC**: 32 bits, the synchronization source identifier for originator of current packet

Above RTCP header is following one or more information packets:

- **RR**: Receiver report, for transmission and reception statistics of not active senders
- **SR**: Sender report, for transmission and reception statistics of active senders
- **SDES**: Source description items, describing session member information
- **BYE**: Indicates end of session member participation
- **APP**: Contains application specific information

(Schulzrinne et al. 2003, date of retrieval 12.1.2014)

The RTP packet format is visually illustrated on *TABLE 1* (*SRTP Master Key Identifier (MKI)* and *Authentication Tag* fields are not included) and RTCP on *TABLE 2* (*E bit*, *SRTCP index*, *SRTCP MKI* and *Authentication Tag* fields are not included) respectively. All fields are in a plain data format being different from the table definitions.

### 2.2.2 Secure RTP

The Secure Real-time Transport Protocol (SRTP) was developed to provide the security services (such as the confidentiality and the integrity for RTP and RTCP packets) for the RTP based audio and video media transmission. Developers behind produced security features were the development teams from Cisco and Ericsson companies. The specification was formally published by the IETF specification RFC 3711. The SRTP extends of the RFC 3550 specification providing an encryption, an authentication and a integrity services for the RTP media stream.

Requirements of the SRTP audio and video packet transmission is a high throughput with low packet expansion. Therefore use of the any additional headers and payload fields need to be

avoided. Due to the header compression mechanism, the RTP header fields need to be left unencrypted and are available for possible eavesdropper. A decryption of encrypted payloads must succeed without some of the datagrams are lost and the decryption of the data stream must be possible to launch before all packets of the stream are received. Due to the previous, cipher requirements created on a cryptographic security services are based on the PRF for the secure key creation, an additive stream cipher for the encryption/decryption, the HMAC for message authentication and packet indexing unit creation for IV's of SRTP/SRTCP (introduced later on this chapter). (Baugher, McGrew, Naslund, Carrara, Norrman, 2004, date of retrieval 26.1.2014; Sisalem et al. 2009, 175-176)

The SRTP itself produces additional services which are increasing security. Firstly, distinct session keys for the confidentiality and the integrity protection of SRTP and SRTCP streams are derived from a single *master key* in a cryptographically secure way. Secondly, the key derivation can periodically refresh session keys. Thirdly, utilize of a salting keys. The master salt key offers protection and additional entropy for the key derivation functionality. Plus, the derived session salt key used in encryption protects against of attacks toward of additive stream cipher. (Baugher et al. 2004, date of retrieval 26.1.2014)

#### 2.2.2.1 SRTP framework

Conceptually, the SRTP can be thought to be a "bump in the stack" implementation which operates as a secure moderator between the RTP application and the transport layer. The RTP packet is modified to the SRTP packet format and sent to a receiver. Respectively, the received SRTP packet is intercepted to the RTP format. (Baugher et al. 2004, date of retrieval 26.1.2014)

The SRTP packet format is illustrated on *TABLE 1*. Single *B* character in the first table row describes bits from 0 until 7 so there are totally 4 bytes in each row. The *P+A* is describing plain (readable) data row which belongs to authenticated portion of RTP packet. The *E+A* is describing encrypted and authenticated data portion. The single *P* character is describing plain data only portion, which may be included to the SRTP packet. The RTP packet fields are introduced in chapter 2.2.1. See of the SRTP additional field descriptions in below.

The SRTP data packet contains following additional, configurable length fields:

- **MKI**: the value is defined and signaled by key management protocol.
- **Authentication tag**: the packet auth. data which is truncated to 32 bit or 80 bit value.

(Baugher et al. 2004, date of retrieval 26.1.2014)

Within the SRTP cryptographic context, a 16-bit rollover counter (ROC) with a concatenation of a 32-bit RTP sequence number (SEQ) is used to define a 48-bit packet index variable (SRTP packet index). The ROC is started from a zero value and it's increased by one whenever sequence number is rolled over. The encryption algorithm may be null (no encryption) or AES in counter mode. The authentication algorithm is HMAC-SHA1. This algorithm produces 160-bit keystream which is truncated to the 80-bit authentication tag by default. Optionally some applications may require shorter (32-bit authentication tag) or even null authentication to be used with the SRTP packets. A null encryption or authentication and short authentication tag (weak authentication) options lead to security risks. The null or short authentication must not be used with the SRTCP packets. The weak authentication is acceptable if small amount of forging is acceptable. Also acceptable when not it's predictable that attacker can modify a ciphertext to be decrypted with intelligent result. For example with the many codecs, for controlled signal manipulation one needs to understand input signal relation to the output signal. The key management protocol produces the MKI length information and a value itself. The MKI may identify particular master key within a SRTP cryptographic context. (Sisalem et al. 2009, 178; (Baugher et al. 2004, date of retrieval 2.2.2014)

*TABLE 1. The structure of the SRTP packet (continues on next page)*

| B | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **P+A** | V | | P | X | CC | | | | M | PT | | | | | | | Sequence number | | | | | | | | | | | | | | | |
| **P+A** | Timestamp | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **P+A** | SSRC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **P+A** | CSRC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **P+A** | RTP Extension (Optional) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A** | Payload | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | |
|---|---|---|---|
| **E+A** | Payload (continued) | RTP padding | RTP pad count |
| **P** | SRTP MKI (Optional) | | |
| **P** | Authentication Tag (Recommended) | | |

The SRTCP packet format is illustrated on *TABLE 2*. Single or first packet of the compound RTCP packet header structure is always left unencrypted. The *E+A** is describing encrypted and authenticated data portion of 2nd until final RTCP packet within compound RTCP packet. Other table definitions can be inspected from the *TABLE 1* introduction. The RTCP packet fields are introduced in chapter 2.2.1. See, description of the SRTCP field additions in below.

The SRTCP data packet contains following additional (new) fields:

- **E-flag**: 1 bit, indicates state of the current SRTCP packet encryption
- **SRTCP index**: 31 bits, counter of an SRTCP packet, starting from zero

(Baugher et al. 2004, date of retrieval 26.1.2014)

Within an SRTCP cryptographic context is shared most of the SRTP parameters except no ROC value is used. The SRTCP maintains a counter item for number of the handled packets for particular master key.

*TABLE 2. The example structure of an (compound) SRTCP packet (continues on next page)*

| **B** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **P+A** | V | | P | | RC | | | | PT | | | | | | | | Length | | | | | | | | | | | | | | | |
| **P+A** | SSRC of sender | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A** | Sender Info | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A** | Report Block 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A** | … | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A** | Report Block n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A*** | V | | P | | SC | | | | PT | | | | | | | | Length | | | | | | | | | | | | | | | |
| **E+A*** | SSRC of CSRC #1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **E+A*** | SDES items | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

| | | |
|---|---|---|
| **E+A*** | | SDES items (continued) |
| **P+A** | E | SRTCP Index |
| **P** | | SRTCP MKI (Optional) |
| **P** | | Authentication Tag (Mandatory) |

#### 2.2.2.2 The key derivation algorithm and PRF

The SRTP endpoints shares secret input variables for a key derivation procedure: denoted as *master key* and *master salt* (both are random variables and salt may be public) via an external key management protocol (such as SDES, MIKEY, ZRTP, DTLS-SRTP). The secret keys in SRTP key derivation are used in two phases as input parameters to the default cipher: AES. In *first phase*; AES with the PRF produces a set of session keys as result of key derivation: an encryption key, a salting key and an authentication key. The SRTP and SRTCP both have an individually derived session keys. The SRTP consist two modes of running AES: AES-CTR and AES in f8-mode. In *second phase*; session keys with data blocks are fed as input parameters to the AES cipher in counter mode. (Sisalem et al. 2009, 176-177, 181-183; Baugher et al. 2004, date of retrieval 26.1.2014)

The SDES and MIKEY can be classified as a signaling plane key management protocols because the SDP attributes are carried in a (signaling) message payloads. Prerequisite from message exchange is complication of a SDP offer/answer model in one round trip. The signaling protocol (ie SIP) payloads must be integrity protected. Differently the ZRTP and the DTLS-SRTP are classified as a media plane key management protocols because (key) message exchange and media exchange are performed concurrently. Scenarios such as forking, retargeting and early media are challenging to the key management protocols. In cases of retargeting and forking the identity of the terminating UAS may be different than originally intended by a callee resulting failure to returning message encryption. In early media, the callee may start sending of media already when the INVITATION request (with SDP parameters) message is received. This includes media clipping when the caller start receiving of media before the SDP answer needed to derive keying material and decrypt receiving message. The early media affects specially for SDES and MIKEY because those requires full round trip for the key derivation. The early media is better handled by

the media plane key management protocols. In these schemes the media is firstly send unencrypted and later changed to the SRTP when the cryptographic context creation are finished by endpoints. (Sisalem et al. 2009, 185-186)

For an example let's consider of two positive integers, named as *m* (for input block size) and *n* (for label). Two input parameter blocks, named as *k* and *x* are placed into the definition of a pseudo-random function family in *FORMULA 3*. Respectively for the given values the secret random key k and m-bit sequence are producing the output value of PRF_n(k,x). Output block is an n-bit sequence, computationally indistinguishable. (Baugher et al. 2004, date of retrieval 1.2.2014)

*FORMULA 3. Definition of a pseudo-random function family (PRF) for set of keyed functions*

$$\mathrm{PRF\_n}(\mathrm{k}, \mathrm{x})$$

For the key derivation procedure of SRTP, a secure PRF with input block size *m = 128* (keyed by 128, 192, or 256 bit master key) must be used. Furthermore, key k_master and IV equal to (x*2^16) are applied with PRF_n(k_master, x) to produce n-bit output blocks (for n up to 2^23). Finally, the produced output block is truncated to the n first (left-most) bits. (Baugher et al. 2004, date of retrieval 1.2.2014)

Key derivation rate is defined for a lifetime of current master key. For calculation process, let's define that *a DIV t* denotes integer division a by t. With non-zero values of t, *a DIV t* can be implemented as a right-shift by base-2 logarithm of t. By default, SRTP uses a key derivation rate of zero. This is interpreted as no key refreshing is used (meaning key derivation shall be taken place only once). In addition of above key derivation rate following items are used in key derivation procedure: a *label* (an 8-bit constant value), master salt key and a unique index variable for the SRTP and the SRTCP in following order.

1) the SRTP or the SRTCP index DIV key derivation rate produces *r* (rate variable)
2) the label || r produces key_id (*key identifier)*
3) the key_id XOR the master salt with the least significant bits produces x variable (a *keystream*) placed to the PRF function.

In the SRTP encryption keystream utilizes label value 0x00 and respectively the SRTCP label value is 0x03. The SRTP salting key label value is 0x02 and respectively the SRTCP label value

is 0x05. Session keys derived from these labels are used for the packet encryption and decryption. The SRTP authentication keystream utilizes label value 0x01 and respectively the SRTCP label value is 0x04. Session keys derived from authentication labels are used to calculate and verify the packet MAC values. (Baugher et al. 2004, date of retrieval 1.2.2014)

#### 2.2.2.3 Packet encryption

Conceptually, the cipher encryption of AES in counter mode is modifying sequence of integers where the starting point of integer sequence is randomized. In the encryption operation, each packet is processed with a distinct keystream block. Produced output keystream shall consist of concatenation of 128-bit blocks. The encryption process is described in *FORMULA 4.*

*FORMULA 4. Encryption of a successive keystream blocks*

$\mathrm{E}(\mathrm{k\_e}, \mathrm{IV}) \,||\, \mathrm{E}(\mathrm{k\_e}, \mathrm{IV} + 1 \bmod 2^{128}) \,||\, \mathrm{E}(\mathrm{k\_e}, \mathrm{IV} + 2 \bmod 2^{128})$ ... where

E = encryption function

k_e = session encryption key, which is 128-bit size sequence of integers

IV = Initialization Vector, which is 128-bit size sequence of integers

(Baugher et al. 2004, date of retrieval 2.2.2014)

The IV for SRTP is defined by placing following variables: the SSRC, the SRTP packet index and the SRTP session salting key to the 128-bit blocks and in mutual relationship between those via the XOR operations. Both of the XOR-sum is padded with necessary leading zeroes to produce 128-bit block as intermediate and as the final result. Corresponding definitions for SRTCP contains the SSRC of the first compound packet header, the SRTCP packet index from 31-bit packet counter with zero prefix bit and SRTCP session keys. The IV block creation is described in *FORMULA 5*.

*FORMULA 5. Creation of encryption IV block*

$\mathrm{IV} = (\mathrm{k_s} * 2^{16}) \oplus (\mathrm{SSRC} * 2^{64}) \oplus (\mathrm{i} * 2^{16})$ where

k_s = session salting key, which is 112-bit size sequence of integers

$\oplus$ = XOR operation

SSRC = the synchronization source identifier

i = SRTP packet index consisting of ROC || SEQ

IV = Initialization Vector, which is 128-bit size sequence of integers
(Baugher et al. 2004, date of retrieval 2.2.2014)

#### 2.2.2.4 Packet authentication

A packet may be stored and altered along transmission path. Replicated packet may be later re-injected to the network. Authentication offers protection of replaying packets and sending manipulated packets to receiver.

The SRTP/SRTCP packets are authenticated (integrity protected) after encryption portion is calculated and placed in. An authentication portion is calculated over source message and truncated digest value is placed to authentication tag field. In the packet receiving side processing order back to the plaintext is performed conversely. The authenticated portion (denoted as *M*) of the SRTP contains header with possible extensions and the encrypted payload section with the concatenation of ROC. On the SRTCP side (concerning both single RTCP and compound RTCP packets) authentication source portion contains all fields except the MKI and the Authentication Tag. (Baugher et al. 2004, date of retrieval 2.2.2014) The HMAC calculation with authentication key and authenticated data is described in *FORMULA 6*.

*FORMULA 6. Authentication keystream blocks*
$\mathrm{HMAC}(\mathrm{k\_a}, \mathrm{M})$ where
HMAC = keyed-Hashed MAC
k_a = session authentication key, which is 160-bit size sequence of integers
M = Authentication portion

### 2.2.3 Key derivation with SIP & MIKEY

The MIKEY key management protocol may be used to perform key exchange and negotiate the security parameters for SRTP. The SRTP is also main use case for the MIKEY although latter is independent secure data protocol. The MIKEY messages (I_MESSAGE & R_MESSAGE) are embedded in the SDP payloads which are encoded in base64 data format. The I_MESSAGE is sent in SDP offer with (SIP) session invitation by an entity called *initiator*. The optional R_MESSAGE is sent back in SDP answer with (SIP OK) session response by another entity

called *responder*. The MIKEY message consists of one header (may be also called to payload) followed by number of actual payload sections. (Sisalem et al. 2009, 191)

In the MIKEY exchange data stream security parameters are carried in a Crypto Session (CS), one or more CSs form a Crypto Session Bundle (CSB). The CS has unique identifier within a CSB which is also identified by unique identifier per initiator and responder pair. For the SRTP CS corresponds to an SRTP cryptographic context. The CSB contents (and thus CSs accordingly) are carried in a MIKEY message header. The header contains also CS ID map which consists of a security policy number, a SSRC and a ROC tuples. (Arkko, Carrara, Lindholm, Naslund, Norrman, date of retrieval 9.2.2014; Sisalem et al. 2009, 193)

In addition to the security policies mapping to crypto sessions a key set is created for each CS. The MIKEY initiator creates a *TEK generation key* (TGK) independently and randomly. A key set consist either a TGK or a Traffic encryption (TEK) with optional Salt which may be utilized by underlying protocol.

Constant prefix label values for key creation from a TGK:

- **TEK**: 0x2AD01C64
- **Salt key**: 0x39A2C14B

The label consists of following concatenation of variables: *a constant key identifier* (Constant values are described above. The SRTP requires TEK (for SRTP master key) and salt key for SRTP's own key derivation procedures. According to the MIKEY specification it's also possible to derive distinct encryption and authentication keys but those are not meaningful for the SRTP because of SRTP has own key derivation process.), *the CS ID*, *CSB ID* and *a random number* (carried in RAND payload). (Arkko et al., date of retrieval 9.2.2014)

#### 2.2.3.1 MIKEY PRF description

The MIKEY key derivation bases on a pseudo-random function (also P-function) PRF(input_key, label) that uses the HMAC-SHA1 algorithm with a input key. Depending on the desired output key the input key can be either a TGK or a pre-shared key PSK (which is introduced in next chapter). The P-function is divided to the inner and outer HMAC transformations where inner gets a key

and a label as input calculation parameters. A key and output of inner operation concatenated with label is fed to outer operation as input parameters. If m variable is equal to 1 then no more data processing is needed. If m variable is greater than one then the HMAC operations are continued as many rounds as m stands for while input parameters are key and output of previous HMAC operation concatenated with label. While input parameter length increases single HMAC output length is always 160 bits. Depending on the m value the partial output values may be concatenated together to get longer final output value. The HMAC operations inside of one P-function are described in *FORMULA 7* on below.

*FORMULA 7. The MIKEY P-Function*

$$\mathrm{P}(\mathrm{s}, \mathrm{label}, \mathrm{m}) = \mathrm{HMAC}(\mathrm{s},\ \mathrm{A}_1 \,||\, \mathrm{label}) \,||\, \mathrm{HMAC}(\mathrm{s},\ \mathrm{A}_2 \,||\, \mathrm{label}) \,||\, \ldots \mathrm{HMAC}(\mathrm{s},\ \mathrm{A}_\mathrm{m} \,||\, \mathrm{label})$$

where

HMAC = SHA-1 based message authentication function

$A_0$ = label

$A_i$ = HMAC(s, A_(i-1))

s = secret key

m = output key length divided by 160, rounded up to the nearest integer

|| = concatenation of two variables

(Arkko et al., date of retrieval 30.9.2014)

The input data ie key content needs to be divided for blocks of 256 bits in MIKEY PRF procedure. For short key lengths only one key block exists. Each block is processed through above P-function (see FORMULA 7) with divided or complete key block and label. Eventually P-output key segments are XOR-operated together to get the final output value. Depending on the desired key length only most significant bits are taken from the output to key value.

*FORMULA 8. MIKEY default PRF*

$$\mathrm{PRF}(\mathrm{inkey}, \mathrm{label}) = \mathrm{P}(\mathrm{s}_1, \mathrm{label}, \mathrm{m}) \oplus \mathrm{P}(\mathrm{s}_2, \mathrm{label}, \mathrm{m}) \oplus \ldots \oplus \mathrm{P}(\mathrm{s}_\mathrm{n}, \mathrm{label}, \mathrm{m})$$

where

$s_1$,$s_2$ = sub-key block of 256-bit

$s_n$= last (and may be only existing) key block of 256-bit or less

label = concatenation of constant and random values where constant values depends on case which invoked PRF

m = output key len divided by 160, rounded up to the nearest integer

⊕ = XOR operation

(Arkko et al., date of retrieval 30.9.2014)

Two following figures: *FIGURE 3. MIKEY default PRF* and *FIGURE 4. MIKEY P-Function* describes and visualizes the output key creation in more detail and those can be used together with latest formulas to familiarize functionality of the MIKEY PRF and the message key creation. The input key is fed through the key splitter which divides the key in blocks of 256-bits which are again processed with a P-Function. Depending from the key length the first or the last input key block may be shorter than 256-bits. All output key blocks from result of P-Functions are XOR-ed together to get the final output value. If only one input key block is used (as happens with relatively short input key values) then no XOR operation is used in the end.

**PRF**

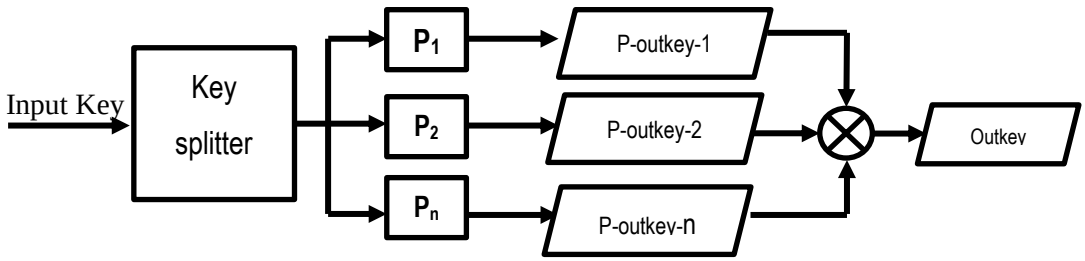



*FIGURE 3. MIKEY default PRF*

(Furht, Kirovski, date of retrieval 15.10.2014)

**P-Function**

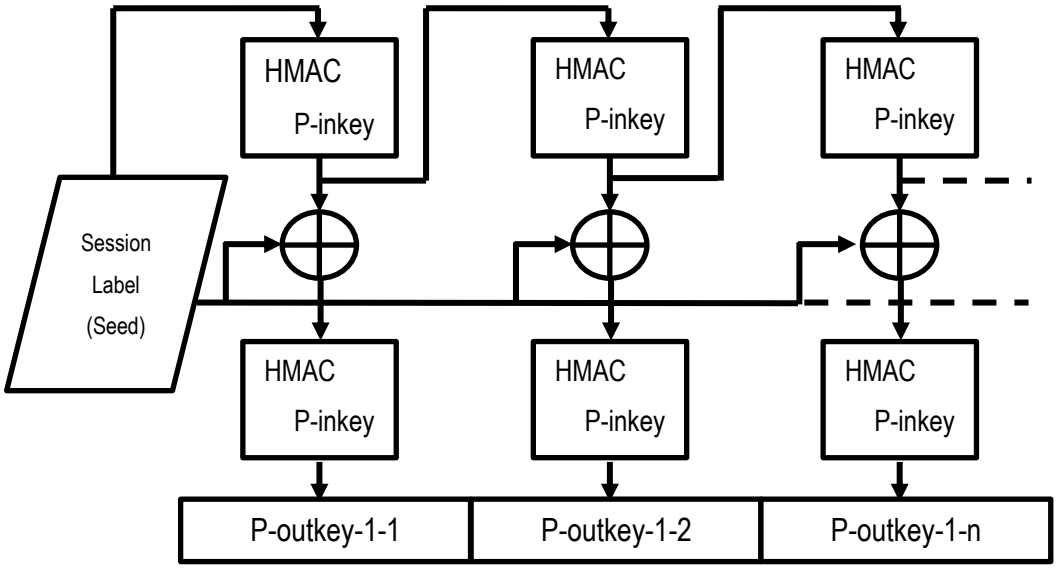



*FIGURE 4. MIKEY P-Function*

(Furht, Kirovski, date of retrieval 15.10.2014)

#### 2.2.3.2 MIKEY key exchange methods

The base version of the MIKEY specifies three following key exchange methods: a pre-shared key (MIKEY-PSK), a public key encryption (MIKEY-RSA) and a Diffie-Hellman (MIKEY-DHSIGN). The first two methods are based on the key transport functionality, where the TGK is sent to the receiver. The latter method exchanges Diffie-Hellman values which are used to derive TGK. From symmetric cryptography point of view, the pre-shared key method is the most efficient way to perform key transport and has an advantage that relatively small amount of the data need to be exchanged. In the next chapters the MIKEY-PSK is studied in more details. Firstly, common header payload is created and following payloads are concatenated to header. Each payload identifies type of following payload by a *Next payload* field. Finally step in the MIKEY-PSK procedure is an authentication with the MAC over complete message and storing of the digest to MAC field of the KEMAC payload. (Arkko et al., date of retrieval 12.1.2014)

Payloads of I_MESSAGE:

- HDR: The general MIKEY header, including CSB data
- T: Timestamp
- RAND: Random or pseudo-random byte-string
- IDx: Identity of initiator (IDi) or Identity of responder (IDr), optional element
- SP: The security policies, one or more entities
- KEMAC: Key data transport

Payloads of R_MESSAGE (optional):

- HDR: The general MIKEY header, including CSB data
- T: Timestamp
- IDx: Identity of initiator (IDi) or Identity of responder (IDr), optional element
- V: MAC calculated verification data

In following *FIGURE 5. MIKEY-PSK message exchange method* is illustrated.

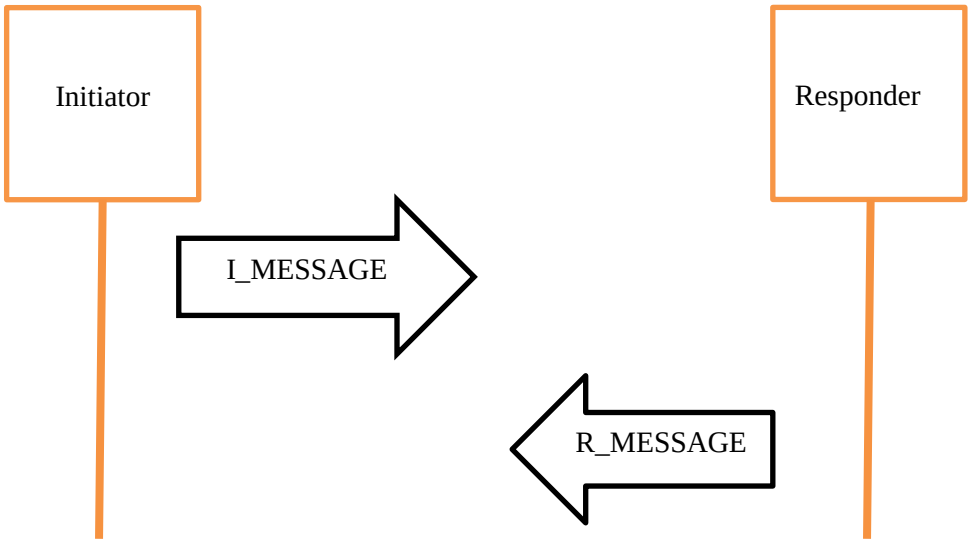



*FIGURE 5. MIKEY-PSK message exchange method*

The SDP attribute *key-mgmt* is used to carry MIKEY key sets in encrypted and authenticated form. A key-mgmt attribute contains a key management protocol identifier which is “mikey” in this use case. Actual MIKEY message is followed after semicolon in a base64 encoded form. The SDP attribute *key-mgmt* is defined either at the SDP session or media level to negotiate and exchange keys. If defined to session level then cryptographic parameters concerns all media

streams that belong to the concerned multimedia session. The media level definition applies only to the RTP session where it is defined and also overwrites possible session level definition. If multiple parameters are defined to same session level then answerer may choose one of those to following (optional) answer. In addition, the SIP signaling itself should be secured to protect the key management messages from a man in the middle attacks. (Sisalem et al. 2009, 201, date of retrieval 25.10.2014)

Key-mgmt attribute (a) follows the ABNF grammar:

- key-mgmt-attribute = key-mgmt-att-field ":" key-mgmt-att-value

The field descriptions shown above are:
*key-mgmt-att-field*: key management attribute identifier defined as "key-mgmt"
*key-mgmt-att-value*: key management protocol identifier defined as "mikey" with white space and concatenated with base64 encoded MIKEY message. For detail descriptions, please see Arkko, chapters 3.1. SDP Extensions. (Arkko, Lindholm, Naslund, Norrman, Carrara, 2006, date of retrieval 25.10.2014)

In following figures *FIGURE 6.* Example of an SDP offer with key-mgmt attribute in media level and *FIGURE 7*. Example of an SDP offer with key-mgmt attribute in session level use of MIKEY media parameters is described. To simplify these use cases, preliminary SDP parameters are omitted.

**SDP offer:**
…
m=audio 49000 RTP/SAVP 98
a=rtpmap:98 AMR/8000
a=key-mgmt:mikey AQAFgM0XflABAAAAAAAAAAAAAAsAyONQ6gAAAAAGEEoo2pee4hp2
UaDX8ZE22YwKAAAPZG9uYWxkQGR1Y2suY29tAQAAAAAAAQAk0JKpgaVkDaawi9whVBtBt
0KZ14ymNuu62+Nv3ozPLygwK/GbAV9iemnGUIZ19fWQUOSrzKTAv9zV
m=video 52230 RTP/SAVP 31
a=rtpmap:31 H261/90000

*FIGURE 6.* Example of an SDP offer with key-mgmt attribute in media level

**SDP offer:**

…

a=key-mgmt:mikey AQAFgM0XflABAAAAAAAAAAAAA…

m=audio 49000 RTP/SAVP 98

a=rtpmap:98 AMR/8000

m=video 52230 RTP/SAVP 31

a=rtpmap:31 H261/90000

*FIGURE 7*. Example of an SDP offer with key-mgmt attribute in session level

The prerequisite for MIKEY-PSK method has achieved when initiator and responder are preconfigured with a PSK. Both MIKEY endpoints have the responsibility to create message keys for encryption and integrity protection via key derivation from shared secret (PSK) in cryptographic secure way. The MIKEY message key derivation process itself is similar to MIKEY key derivation from TGK introduced earlier chapters. Label consists of concatenation of following variables: a constant key identifier, a fixed parameter 0xFF, CSB ID and a random number (carried in RAND payload). (Sisalem et al. 2009, 195-196; Arkko et al., date of retrieval 9.2.2014)

Constant values for message key creation from a PSK:

- **Encryption key**: 0x150533E1
- **Authentication key**: 0x2D22AC75
- **Salting key**: 0x29B88916

The content of the TGK is determined by initiator and sent to the responder inside of I_MESSAGE. The TGK is encapsulated and encrypted as sub-payload to the KEMAC payload. The KEMAC is MIKEY key encrypted data block which contains MAC calculated over encrypted initiator message and result of the authentication digest placed to end of the KEMAC payload. The Formula 9 defines KEMAC payload creation. (Arkko et al., date of retrieval 11.2.2014)

*FORMULA 9. KEMAC sub-payload encryption and concatenation of authentication digest*

$\mathrm{KEMAC} = \mathrm{E}(\mathrm{M_e}, \{\mathrm{TGK}\}||\mathrm{MAC})$ where

E = encryption function

$M_e$ = MIKEY message encryption key, which is 128-bit size sequence of integers

TGK = TEK generation key

{..} = zero or more items

MAC = authentication digest

Essential for the MIKEY key exchange, next construction of the KEMAC payload data structure is studied in more details. In the MIKEY-PSK case, KEMAC payload must be last payload of message and it's signaled by value of zero in Next payload field. The encryption algorithm describes method used encrypt data of key data sub-payloads. The selected method is AES-CTR using a 128-bit encryption key. The authentication algorithm is HMAC-SHA1. (Arkko et al., date of retrieval 16.2.2014)

*TABLE 3. The key data transport payload (KEMAC)*

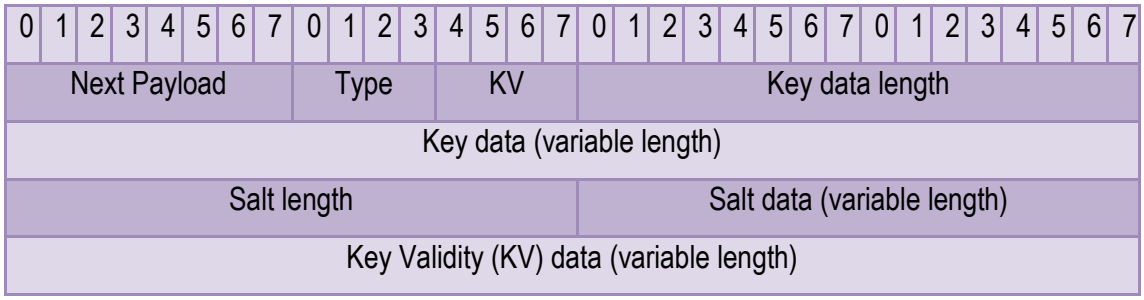
| 0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 | 0 1 2 3 4 5 6 7 |
|---|---|---|---|
| Next Payload | Encryption Algorithm | Encryption data length | |
| Encrypted data (key data sub-payloads, variable length) | | | |
| MAC Algorithm | MAC (digest, variable length) | | |

The key data payload contains keying material for the SRTP connection. Type field indicates the key set included. The TEK is useful and recommended when pre-encrypted material exist before MIKEY keying. The KV field indicates type of key validity period (SPI/MKI is defined by 0b0001, Interval is defined by 0b0010) and signals to a SRTP cryptographic content state of MKI indicator (see actual data definition in below). Salt (the SRTP Master salt) is included if defined in Type field. (Arkko et al., date of retrieval 16.2.2014)

The key data sub-payload type field definitions (in bit values):

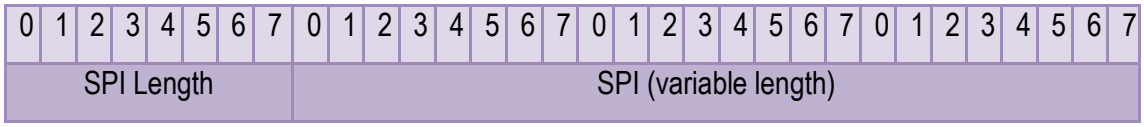
- **TGK**: 0b0000
- **TGK+Salt**: 0b0001
- **TEK**: 0b0010
- **TEK+Salt:** 0b0011

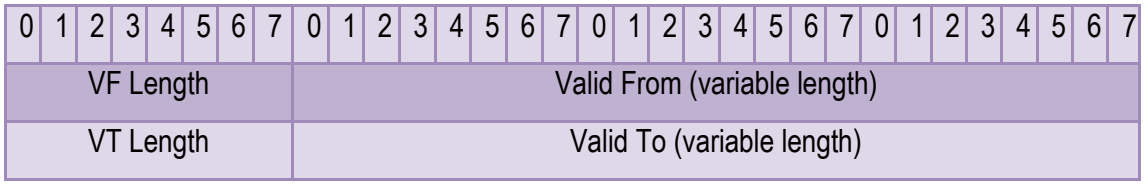
*TABLE 4. The key data sub-payloads*




The key validity (KV) data is used to specify the SRTP MKI value. The KV itself is not a standalone payload but instead sub division of key data sub-payload. The value of KV may be null when it is not included. (Arkko et al., date of retrieval 16.2.2014)

*TABLE 5. The KV data field SPI and Interval definitions based on value of KV field.*




Sub-payload fields of the KEMAC payload are encrypted with AES algorithm in counter mode. Initialization vector for encryption is defined in Formula 9. The MIKEY-PSK may be used with null encryption and authentication if security can be guaranteed by SIP (the SDP packet need to be encrypted end-to-end). (Arkko et al., date of retrieval 16.2.2014)

*FORMULA 9. Creation of KEMAC encryption IV block*

$\mathrm{IV} = (\mathrm{M_s} \oplus (0\mathrm{x}0000 \,||\, \mathrm{CSB\ ID} \,||\, \mathrm{T})) \,||\, \mathrm{ctr\_value}$ where

IV = Initialization Vector, which is 128-bit size sequence of integers

$M_s$ = MIKEY message salting key, which is 112-bit size sequence of integers

$\oplus$ = XOR operation

CSB ID = Crypto Session Bundle ID, 32-bit

T = timestamp of T payload, 64-bit

ctr_value = increased counter, started from zero

### 2.2.4 Key derivation with SIP & SDES

Like the MIKEY also the SDES offers method to negotiate security key parameters for use of security protocol such as the SRTP. Keying material of the SDES is transferred using *crypto* parameter in a SDP payload. The crypto attribute is encoded in base64 data format and negotiated using a SDP offer/answer model (also similar than in the MIKEY). The main difference between these two key management transferring methods is that crypto parameter is carried in a cleartext form in SDP payload. Therefore the SDES needs encryption and authentication services from transmission protocol (the SIP signaling). (Andreasen, Baugher, Wing, Session Description Protocol (SDP) Security Descriptions for Media Streams. Date of retrieval 15.10.2014)

The SDES crypto attribute is used only at SDP media level to describe cryptographic suite, key and session parameters for ongoing media session or for one being to establish. Crypto attribute (a) follows the ABNF grammar:

- attibute = crypto:<tag> <crypto-suite> <key-params> [<session-params>]

The field descriptions shown above are:

*tag*: decimal number which is an unique identifier for certain media attribute definitions

*crypto-suite*: specified identifier for encryption and authentication algorithms protecting media stream. For detail descriptions, please see Andreasen, chapters 6.2. Crypto suites.

*key-params*: specifies set of keying material with following structure:

- key-params = <key-method> ":" <key-info>
- inline: base64(<key> || <salt>)["||"<lifetime>] ["||"<MKI>":"<MKI-length>]

where *inline* is used as key-method indicator followed by semicolon and actual keys itself. With the SRTP key-info contains concatenation of the master and the salt keys and latter result in the base64 encoded form. The key-info line may also contain two more optional parameters: *lifetime*, which defines how long keys are enable to use and *master key identifier (MKI)* which associates certain master key to the incoming SRTP packet. If lifetime parameter is missing then following

default values are assumed; $2^{48}$ for the SRTP packets and $2^{31}$ for the SRTCP packets. If the MKI is given then length in bytes is must be followed after semicolon.
*session-params*: contains an additional SRTP cryptographic context parameters. For detail descriptions, please see Andreasen, chapters 6. SRTP Security Descriptions.
(Andreasen et al., date of retrieval 15.10.2014)

A SDP offer or proposal sent to answerer may contain several crypto attribute definitions for media lines. Answerer accepts one among of all crypto parameters (if any) and attach accepted with updated key values to the SDP answer. After receiving of the SDP answer, the initiator may use received key set to protect the SRTP media stream. Receiver is capable of decrypting the SRTP packets by using master keys known only by media session counterparts.

In following *FIGURE 8*. Example of SDES offer/answer use of the SDES media parameters is described. To simplify this use case, preliminary SDP parameters are omitted.

**SDP offer:**

…

m=audio 49170 RTP/SAVP 0

a=crypto:2 AES_CM_128_HMAC_SHA1_80
inline:WVNfX19zZzW1jdGwgKCkgewkyMjA7fQp9CnVubGVz|2^20|1:4;
inline:WKMbsd5KDa8mEond4QsbBa92akDFjfjAJFKsdJHjfeY|2^20|2:4

**SDP answer:**

…

m=audio 32640 RTP/SAVP 0

a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:PS1uQCVeeCFCanVmcjkpPywjNWhcYD0mZZtxeVBR|2^20|1:4

*FIGURE 8*. Example of SDES offer/answer

# 3 CREATION OF SECURE RTP FUZZING TEST SUITE

In a model based fuzzing a protocol specification is modeled in principles of a context-free grammar called Backus-Naur Form (BNF). The Codenomicon Defensics native grammar tree format (.bbt) is used in the most cases to define protocol model. In addition, the Augmented Backus-Naur Form (.abnf) or the Abstract Syntax Notation One (.asn) may be used to update a model definition. A protocol modeling begins from a start or root symbol. The implemented model is not full protocol implementation but it describes a protocol specification and the message exchange from black box testing aspects. A test suite file is XML structured file with Codenomicon Defensics .bbx file format extension. The test suite file is used to load protocol model (one or more .bbt file(s) with test cases and anomaly library) in a development phase. A final test suite component is usable after installation of the Java file (.jar). The test suite loads also rule definitions and user settings to the model. Use of rule adds dynamic content handling and calculations to model. One special group of rules is defined by input/output (IO) rules which provide test case delivery to the SUT. Target parameters and protocol configuration are processed by user settings. Outgoing and incoming message handling functionality of the test suite is in most cases implemented by suite specific Java package(s). The Eclipse programming environment is used in a Java software development.

A test suite is implemented with the valid protocol functionalities. The suite is acting as valid protocol entity. Suite's role is a *client* when it mainly receiving messages or a *server* when it's sending messages to the SUT. A valid case is error-free protocol implementation. Valid case is modified with *an anomaly* to form of a *test case*. Test cases are sent as messages to the SUT while traversing protocol tree structure.

A simplified description of the test case run is firstly to send a test case (anomaly) and secondly verify operation of SUT with sending of a valid test case that except response from the SUT. This operation is continued until all test cases are passed or an error situation is encountered. An error might appear when tested application is crashed after receiving of test case or is not able to respond valid requests anymore.

### 3.1 Modeling of SRTP Fuzzer Test Suite

The black box model of the RTP consists of *rtp-suite.bbx* and *rtp-model.bbx*. Earlier defines name and version of the test suite, anomaly library used to produce anomalized messages of protocol description. It also makes base groups defined by a test tool developer to divide test cases to the logical groups. The latter imports user settings, suite and object rule definitions. It also parses all added protocol description files (*.bbt). The RTP and RTCP packet structures with secure extensions from RFC3550 and RFC3771 are defined to *model.bbt* and it's secure extension *secure-rtp.bbt* protocol modeling file. Boolean user setting *sel-srtp* from *settings.bbx* launches creation of the secure rtp protocol leaf when enabled by user of the test suite. A type rule instance from rule library called *type-srtp* creates then the SRTP or the SRTCP model leafs to messages which are sent to a target application. User settings are accessible before running of the test suite via Protocol Monitor (PM) user interface or via command line. Below is shown pieces of the model where user settings and protocol model description are converged to enable secure rtp leaf modeling.

settings.bbx:

```
<setting name="sel-srtp">
  <group>RTP Settings[SRTP Options]</group>
  <description>Secure RTP Selection</description>
  <command-line true="—enable-srtp" false="—disable-srtp"/>
  <default-value>false</default-value>
</setting>
```

rules.bbx:

```
<!—Type rules used to control srtp fields - ->
<rule built-in="type" name="type-srtp">
  <setting name="sel-srtp" param="set"/>
</rule>
```

Please, see also related modeling of the secure-rtp.bbt from appendix 2.

The MIKEY protocol model with default initiator message from RFC3830 is defined to the mikey.bbt suite modeling file. The mikey.bbt defines the most of the protocol contents but the TGK/TEK keys and the salt key (TEK and salt is used as SRTP master key and salt respectively), possible MKI value and authentication signature to be attached end of the MIKEY-I-Message.

### 3.2 Java package design for SRTP Fuzzer Test Suite

A message handling functionality is implemented to a *test run controller* component. It responsibility is to send and receive test cases produced by the RTP suite to the SUT over Internet connection. The *Message API* service offered by platform engine is used to get access and modify messages. Overloaded engine methods to be run in normal suite operation are *initialize* method which is called in the start of the suite test run, *runTestCase* method which is called in the begin of every test case, *evaluate* method may be used to handle suite trigger calls from model. The suite triggers contains data of a current tree branch or a leaf. The use of these triggers gives developer a way to modify sent messages to contain for example the encryption data or authentication signatures. In examples below, the simplified data flows for encryption and authentication of the RTP packets are presented. The SRTP suite trigger definition and implemented controller sub-packages are used for cryptographic calculations. Please, see first sample of the rtp packet modeling from appendix 1.

Encryption and authentication trigger handlings

1) A rule trigger definitions in model => !rtp:encrypt (symbol), !rtp:authenticate(symbol)

2) The controller encryption implementation => evaluate(trigger, symbol..) Firstly received symbol is evaluated and encryption trigger is identified. Enc. trigger is handled first because it's inner definition in a tree structure (and for a reason it's specified an encryption is calculated always before an authentication calculation over modified data). The sequence and plain payload values are evaluated and encoded to byte array value.

3) The controller encryption sub-package implementations => A RTP packet modifications to the SRTP packet for example adding an encryption and an authentication to source packets (RTP) using the Java crypto class package (javax.crypto package provides the classes and interfaces for cryptographic operations). The packet handling requests the AES cipher object from the keyManager object of RTPKeyManager class and forwards plain data packet to an appropriate sub-package class. The encrypted data packet is received as return value and set to the RTP-Payload sym-

bol using the MessageAPI method calls. In this phase, the encrypted symbol value is returned from evaluate() method.

4) The controller authentication implementation => evaluate(trigger, symbol..) Again, firstly a received symbol is evaluated and received authentication trigger is identified. Next phases are done in a private authenticateData() method; authentication tag length is solved from the suite user settings, correct authentication tag symbol is set using MessageAPI interface, data from the authentication source symbol is transferred to a byte array, keyManager object is used to authenticate source data and truncate authentication result to the desired length. The authentication tag is set in place to the end of message. Final step in authentication is to change result message to a symbol via MessageAPI. Edited symbol is next given to a method which checks has MKI value in use and sets it in place if necessary. After this, the secure rtp packet is ready and evaluated symbol of it is returned from evaluate() method.

5) The controller send() method transmits valid or anomaly SRTP packet to SUT.

There are suite triggers and trigger handling implementation in controller also for the SRTCP compound packets. The packet building in this case is a little more multilateral since 1 to n SRTCP packets are sent in one message transmission. The first packet header section is plain (ie visible) while it's payload is encrypted likewise all following packets (header and payload) are encrypted. The individual SRTCP packet has also length and counter rules which must be evaluated before all packet data is available. All the data to be encrypted is collected until a special end trigger (from model) has received which flags from situation where all data is ready to be encrypted. After the encrypted data is stored to the message then authentication needs to be done over combination of the plain and the encrypted data with the SRTCP encryption bit and the SRTCP counter fields. The authentication tag is then placed to the end of message like in case of composing the SRTP packet.

There are situations when another protocol services are required. One such case is to do the *session negotiation* with embedded key management protocol between the signaling protocol (for example SIP) and the RTP target application (also SUT in fuzzing purpose). The session negotiation phase is performed in suitable place of *runTestCase* method before sending of the actual test case. This phase contains also run of the key management services via sub-classes if the STRP settings are selected and activated with selection of the one available key management protocol. The utilized Codenomicon controller session negotiation concept is called later on as a Master-Slave. In this case, the RTP suite role is master and the SIP suite role is slave. The master suite

initializes session parameters and sends those in web service (HTTP) request(s) to the slave suite to do session negotiation or to request other services. When a media session is established between two parties (in this case between slave suite and SUT), the session connection parameters are stored to XML format. Reading of the XML parameters is available for session originator ie Master suite. The master suite concludes session to be activated when owned XML reader object's session parameters differ from a null value where it was firstly initialized. This state enables call of the IO rule initialization method *initRules()* where output rule of the RTP suite is updated with negotiated session parameters. In the begin of *initRules()* also validity of a key management responses in case of the SDES and the MIKEY are checked via keyManager object. In case of valid and changed key management response from previously proposed request then changes are stored to the model and the appropriate cipher objects in controller. In case of error test suite exception is thrown which stops run of test case. In addition, one important task of the slave suite is to maintain activity (ie alive) state of session. The master suite gets regularly updates of session state via these alive-requests. If session is concluded to be terminated and it can't be re-established anymore then master suite signals failure to user as main log entry. The slave suite writes session parameters to the output stream according to received response from SUT. The master suite uses received session parameters to send RTP/SRTP packets to the SUT. Master suite should also observe state of session by the means specified protocol is supporting. That is needed for the reason because RTP is usually implemented on top of a UDP and thus can't easily recognize for example malfunction of a target application.

The implemented controller package may be run to send RTP/SRTP packets to configured target directly (so called "pure RTP" mode) or after receiving media target parameters using a session negotiation via a signaling suite with the target SUT. In pure RTP mode, to be able use of the secure RTP in effectively setting of the master key to suite and to the SUT are required by user.

Secure components from the Rtp java packet:

- SecureRTPAlgorithmBase.java (Base class)
- SecureRTPAES.java (Subclass)
- SecureRTPMikey.java (Subclass)
- SecureRTPSDES.java (Subclass)

The created cipher object is instance of a SecureRTPAlgorithmBase. A subclass definition extends base class implementation. When a new subclass object is created then also base class has to be initialised. The base class has different constructors purposed for object initialization.

### 3.2.1 Class SecureRTPAlgorithmBase

The SecureRTPAlgorithmBase is abstract (base) class which implements constant definitions in CryptoConstants class.

- **constructors**

public SecureRTPAlgorithmBase(byte[ ] keyPriv, byte[ ] keySalt, int iEnc, int iAuth)

- For AES object initialization

public SecureRTPAlgorithmBase(byte[ ] keyPSK)

- For MIKEY object initialization

public SecureRTPAlgorithmBase(boolean isRandom, int iEnc, int iAuth)

- For SDES object initialization

- **method definitions**

public void initializeCipher(byte[ ] keyPayload_upper, byte[ ] keyPayload_lower, byte[ ] timeStamp)
public void initializeCipher(boolean isSRTPMode)
public abstract byte[ ] handlePacket(byte[ ] plainData, byte[ ] rtpSSRC, int iSeqNbr,
byte [ ] srtcpInd)
public abstract byte[ ] authenticatePacket(byte[ ] pdu, int iTagLen)
public abstract byte[ ] getMasterKey()
public abstract byte[ ] getSaltKey()
public abstract void updateKeys(byte[ ] newMasterKey, byte [ ] newSaltKey)
public void initAESCounter(byte upperCounter, byte lowerCounter)
public abstract void reset()

### 3.2.2 Class SecureRTPAES

The SecureRTPAES class extends SecureRTPAlgorithmBase and implements RTPConstants class.

- **constructors**

public SecureRTPAES()
public SecureRTPAES(int iEnc, int iAuth, byte[ ] privKey, byte[ ] saltKey)

- **overridden (base class) methods**

public void initializeCipher(byte[ ] keyPayload_upper, byte[ ] keyPayload_lower, byte[ ] timeStamp)
public byte[ ] handlePacket(..)
public byte[ ] authenticatePacket(..)
public byte[ ] getMasterKey()
public byte[ ] getSaltKey()
public void updateKeys(..)
public void initAESCounter(..)
public void reset()

- **protected or private methods**

protected void createRawKey(byte[ ] keyCipherInput, byte label, int incCounter)
private void initCipher()
private void prepareIv(boolean updRTP)
private void deriveAuthKey(boolean isSRTPMode)
private byte[ ] transformPacket(byte[ ] dataPayload, byte[ ] rtpSSRC, int iSeqNbr, byte[ ] srtcpInd)

The SecureRTPAESTest.java is used to run unit tests for implementation of class SecureRTPAES. Test vectors were created from document sources of Dworkin, Special Publication 800-308A: 2001. chapter F.5 CTR Example Vectors, pages 55-56 and of Baugher, RFC3711 chapters B.2. AES-CM Test Vectors and B.3. Key Derivation Test Vectors, pages 51-53. Documented test vector sources were compared to results in an encryption, a decryption and an authentication of packets. Unit test results were corresponding to the results of source documents.

### 3.2.3 Class SecureRTPMikey

The SecureRTPMikey class extends SecureRTPAlgorithmBase and implements RTPConstants class.

- **constructors**

public SecureRTPMikey()
public SecureRTPMikey(byte[ ] mikeyPSK)

- **overridden (base class) methods**

public void initializeCipher(boolean isSRTPMode)
public byte[ ] handlePacket(..)
public byte[ ] authenticatePacket(..)
public byte[ ] getMasterKey()
public byte[ ] getSaltKey()
public void updateKeys(..)
public void initAESCounter(..)
public void reset()

- **protected or private methods**

protected void createRawKey(byte[ ] keyCipherInput, byte label, int incCounter)
private void initCipher()
private void prepareIv(boolean initIV)
private int getNumberOfHMACIterations(int outKeyLen)
private byte[ ][ ] splitInKey(byte[ ] inKey)
private void doKeyDerivation()
private byte[ ] prfMikey(byte[ ] inKey, byte[ ] keyLabel)
private byte[ ] [ ] p_HashFunction(byte[ ] secret, byte[ ] sesLabel, int numberOfIter)
private void createTGK()
private byte[ ] createMIKEYKeyLabel(String strKeyConst, Boolean tgkKey)
private void cutKeyResult(String keyPrefix, byte[ ] keyRawData)
private void clearVar()

The SecureRTPMikeyTest.java is used to run basic unit tests for implementation of class SecureRTPMikey. There were no test vectors available for verify key creation and cipher or authentication functionality.

### 3.2.4 Class SecureRTPSDES

The SecureRTPSDES class extends SecureRTPAlgorithmBase and implements RTPConstants class.

- **constructors**

public SecureRTPSDES()
public SecureRTPSDES(boolean isRandom, int iEnc, int iAuth)

- **overridden (base class) methods**

public void initializeCipher(boolean isSRTPMode)
public byte[ ] handlePacket(..)
public byte[ ] authenticatePacket(..)
public byte[ ] getMasterKey()
public byte[ ] getSaltKey()
public void updateKeys(..)
public void initAESCounter(..)
public void reset()

- **protected or private methods**

private void createKeySecrets()
private void authenticateSecrets()

Unit test class was not created for SDES. The SecureRTPSDES.java simply creates keys to be used with the signaling protocol SIP and the SDP Security Descriptions as key management protocol. The RTP controller checks key validity from the session response and gives warning if keys are not acceptable or if the MKI is missing when it was requested. The SDES key management functionality was tested with the SFLPhone as negotiator and actor of the SRTP SUT.

# 4 CONCLUSIONS AND DISCUSSION

This project was versatile learning experience for many technology areas which was not well mastered by the author beforehand. One side of the project was an interesting and equally challenging journey to the world of cryptography. To studying, examining and familiarizing the ciphers functionality, authentication principles took some time more than was originally planned. Also it took some amount of time to understand profoundly of AES / AES-CTR encryption and HMAC SHA-1 authentication calculations and for example an idea how PRFs are working in generally by studying RFC documentation and NIST specifications.

The target applications for testing were run on virtual machines. The Oracle VirtualBox was used to setup virtual machine environment and the Linux based Ubuntu and Lubuntu were used as operation systems. Some difficulties were faced when installing and configuring applications to be used for testing. Library dependencies in making and in installing software caused conflicts during the process which forced to look for help from the Internet. This is quite common in Linux environments where software releases appears fairly often and then guides and instructions get obsoleted for the new version of operation system and application versions available with it. Author had only some previous experience of Linux systems so this was very educational also in that sense.

There were some complexity and challenges also with the MIKEY used as the key management protocol. To understand of how both the MIKEY PRF and how the HMAC authentication are processed and calculated needed careful studies and thinking because specification approach is planned and targeted to be in quite universal level for protocols using the MIKEY as key management although the SRTP was targeted to be main use case of a MIKEY. The MiniSIP as target software for the MIKEY is open source product. It was made and developed by group of students in Swedish university but which activity has stopped many years ago. There is one clear error in the MiniSIP software which noticed during development. According to Arkko, chapter 5. Example Scenarios. (Arkko et al., date of retrieval 21.11.2014) one specified alternative is that the MIKEY line can be placed to session level which then protects all underlying media level descriptions. Other option is to place the MIKEY line to one specific media level description which then protects only concerning video or audio description. When the MIKEY line was placed to media level then the MiniSIP application accepts request with OK answer without checking the

given MIKEY parameter. Parameter is supported in session level better but authentication did not get accepted despite of careful checking of model and controller implementation. There are two remaining possibilities of an authentication error. It's either an error in author's MIKEY implementation because two MiniSIP implementation works when those are set to communicate with each other or there is implementation mistake hiding in the MiniSIP implementation which is not detected yet. But either possibility has not proven to exist.

Currently testing of the SRTP tool is about to finish while this MIKEY authentication issue one bigger problem which has not been resolved yet. Several test applications were used to test how the SRTP tool can cope with different kind of use scenarios. The testing phase is very important and useful since not all use cases are not possible to see or think through beforehand. During the testing many errors were found and fixed to product and thus released tool is much more robust product. Software used as target applications were:

- Twinkle -> for SIP session negotiation and for RTP packets no SRTP support
- SJPhone -> for SIP session negotiation with SDES key management and for SRTP packets
- MiniSIP -> for SIP session negotiation with MIKEY key management and for SRTP packets

In this work was concentrated mainly to use the SIP suite as primary slave suite for the RTP/SRTP suite as master suite. The SIP suite can be run either stand-alone (ie normal) or slave mode. All SIP side implementation has been omitted in this thesis work report to keep report size in reasonable level. In the next this effort is mentioned in few words. Use of SIP as slave suite has required a quite amount of SIP model studying and doing of required model additions. Also significant amount of a Java programming and debugging have been done to the SIPSlaveService class (which could be called as Master/Slave interface to SIP Java packet), the SIPRule class (main class of SIP), the SIPRegistration class and the SIPDigestAuthentication class. During the work, the SIP implementation and functionality is become more and more familiar and now doing of changes SIP implementation is now much more smoother by author when compared to early days in this task. Also later on it was considered with other tool developers to separate SIP slave from stand-alone functionality to ease of making changes either one of functional entities and not making of defect by mistake to the other entity.

Further development of the SRTP (so called re-organization) is planned with management of the company and implementation work is started by author. The product is currently based on sequence files (.seq) which are editable by user and to capture files (.pcap) of real media data transfer or streaming traffic. In the near future product is planned to be changed use only capture files of real traffic to make valid and fuzzed (S)RTP test cases based on loaded traffic capture file. The product will offer default media clip in different formats or user can load on he/she own media clips used for testing to target suite. Regardless of the loaded media clip the test case will begin with (S)RTCP Receiver Report (RR) with SDES packet and contain next burst of (S)RTP packets based on loaded media content. The (S)RTCP packets are defined to model and attached to send test cases in controller. Selection of the included (S)RTCP packets can be controller from user interface. Also number of new or changed user settings will be introduced. Some of these are introduced below.

- Number of burst packet can be controlled.
- SSRC value with fixed default value for use of RTP stream and also possible set by user
- Payload type can be selected from predefined numerical values and in case of dynamical payload type also payload format can be selected and/or defined new format.
- Packet delay (between sent packets) setting has default value which may be overridden.

Support for the Session Announcement Protocol (SAP) is needed to add for product to interoperate with some group of streaming applications. Sending of the SAP announcement packet before the RTP test cases opens channel (address and port) from target application to receive (S)RTP media traffic which is fuzzed test cases in this use case. Use of the SAP announcement option will be controlled by user setting.

## REFERENCES


Andreasen, F., Baugher, M., Wing, D. (Cisco Systems) 2006. Session Description Protocol (SDP) Security Descriptions for Media Streams. Date of retrieval 16.3.2014.
http://tools.ietf.org/html/rfc4568

Arkko, J., Carrara, E., Lindholm, F., Naslund, M., Norrman, K. (Ericsson Research) 2004. MIKEY: Multimedia Internet KEYing. Date of retrieval 11.1.2014
http://tools.ietf.org/html/rfc3830

Arkko, J., Lindholm, F., Naslund, M., Norrman, K. (Ericsson), Carrara, E. (Royal Institute of Technology) 2006. Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP). Date of retrieval 25.10.2014
http://tools.ietf.org/html/rfc4567

Baugher, M., McGrew, D. (Cisco Systems, Inc.), Naslund, M., Carrara, E. & Norrman, K. (Ericsson Research) 2004. The Secure Real-time Transport Protocol (SRTP). Date of retrieval 3.11.2013
http://tools.ietf.org/html/rfc3711

Federal Information Processing Standards (FIPS). Publication 197: 2001. Advanced Encryption Standard (AES) (FIPS PUB 197). Date of retrieval 25.1.2014.
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf

Furht, B., Kirovski D. Multimedia Encryption and Authentication Techniques and Applications. Date of retrieval 15.10.2014
http://books.google.fi/books?id=NDA14SqLmecC&pg=PA185&lpg=PA185&dq=mikey+prf&source=bl&ots=L1OUpmjjhb&sig=CViELLc1NjLxIUJkf_OF06lpj6E&hl=fi&sa=X&ei=2kw9VM-NEuv4yQOvh4HoCQ&ved=0CCgQ6AEwAQ#v=onepage&q&f=false

Dworkin, M. National Institute of Standards and Technology (NIST). Special Publication 800-308A: 2001. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. Date of retrieval 25.1.2014.

http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

Handley, M. (UCL), Jacobson, V. (Packet Design), Perkins, C. (University of Glasgow) 2006. SDP: Session Description Protocol. Date of retrieval 16.3.2014.
http://tools.ietf.org/html/rfc4566

Kaksonen, R. 2001. A Functional Method for Assessing Protocol Implementation Security. Technical Research Centre of Finland. VTT Publications 448. Licentiate thesis. Date of retrieval 5.11.2013. http://www.vtt.fi/inf/pdf/publications/2001/P448.pdf.

Porter, T. 2006. Practical VoIP Security. Rockland: Syngress Publishing.

Ransome, J. & Rittinghouse, J. 2005. VoIP Security. Burlington: Elsevier Digital Press.

Schulzrinne, H. (Columbia University), Casner, S. (Packet Design), Frederick, R. (Blue Coat Systems Inc.) & Jacobson, V. (Packet Design) 2003. RTP: A Transport Protocol for Real-Time Applications. Date of retrieval 3.11.2013
http://tools.ietf.org/html/rfc3550

Sisalem, D., Floroiu, J., Kuthan, J., Abend, U., Schulzrinne, H. 2009. SIP Security. West Sussex: Wiley, J. & Sons Ltd.

Stevens, M. 2012. Cryptanalysis of MD5 & SHA-1. Date of retrieval 25.1.2014.
http://2012.sharcs.org/slides/stevens.pdf

Takanen, A. & Vuontisjarvi, M. 2011. Ensuring a Secure Enterprise VoIP Solution. Date of retrieval 3.11.2013
http://www.codenomicon.com/resources/whitepapers/Codenomicon_wp_Ensuring_secure_enterprise_VoIP_solution_20110420.pdf.

Takanen, A. & Juuso, A-M. 2010. Unknown Vulnerability Management. Date of retrieval 11.1.2014

http://www.codenomicon.com/resources/whitepapers/codenomicon-wp-unknown-vulnerability-management-20101019.pdf

Codenomicon. 2014. Date of retrieval 6.1.2014.
http://www.codenomicon.com/products/buzz-on-fuzzing.shtml

**RTP PACKET MODELING** APPENDIX 1

Here is sample of RTP packet modeling. RTP-Packet is root symbol. !rtp* are trigger definitions. [xy] is how optional symbol is defined.

```
# RTP Packet format
RTP-Packet = !rtp-authenticate (
  auth-source
  SRTP-Trailer
)
auth-source = (
  RTP-Header
  rtp-encryption
)
rtp-encryption = !rtp:encrypt (
  RTP-Payload
  [PaddingOctets]
)
```

Common modeling (common.bbt)

```
bit = (0b0 | 0b1)
octet = 0x00-0xFF
```

Example of Secure RTP modeling (secure-rtp.bbt)

```
SRTP-trailer = (
  !type-strtp:get [srtp-elements]
)
SRTCP-fields = (
  !type-strtp:get [srtcp-fields]
)
SRTCP-trailer = (
  !type-strtp:get [srtp-elements]
)
# SRTP lower elements
srtp-mki-value = 1..n(octet)
srtp-mki = [srtp-mki-value]

# OPTIONAL MKI and auth
srtp-elements = (
  .MKI:                 srtp-mki
  .Auth-tag:            auth-tag
)
# SRTCP lower elements
srtcp-fields = (
  .E:                   e-bit-value
  .SRTCP-Index:         srtcp-index-selection
)
e-bit-value = (!type-cipher:get(0b0|0b1) | bit)
srtcp-index-selection = (!srtcp-index 31(bit) | bit))
```

```
auth-tag = !srtp-auth-tag:get (
    .Auth-Tag-32:          auth-tag-32
  | .Auth-Tag-80:          auth-tag-80
)
auth-tag-32 = 4(octet)
auth-tag-80 = 10(octet)
```